Getting Started with Laravel: A Beginner-Friendly Guide

Laravel is a tool that helps you build websites and web applications easily. It is based on PHP, a programming language used to create dynamic web pages. This guide will explain everything step-by-step, using simple words, so you can follow along even if you're new.

What is Laravel?

Laravel is a framework. Think of it like a box of tools and instructions that helps you build a website faster and better. For example:

- Instead of writing long and repetitive code, Laravel gives you shortcuts.
- It organizes your project so your code is clean and easy to manage.
- It includes built-in tools for common tasks like user login, database management, and sending emails.

Why Use Laravel?

If you’ve worked with plain PHP before, you know it can get messy as your project grows. Laravel solves this by:

- Keeping your code organized with clear structures.
- Saving you time with ready-to-use features.
- Making your project easier to maintain and secure.

What Do You Need to Start?

Before using Laravel, make sure you have:

1. **PHP Installed**:
   - PHP is the programming language Laravel is built on. You need version 8.0 or higher.
2. **Composer**:
   - Composer is a tool that helps you install Laravel and manage its dependencies (extra tools Laravel needs to work).
3. **A Text Editor**:
   - Use a program like Visual Studio Code to write your code.
4. **A Database**:

    - A database stores your website's information, like user accounts or product details. MySQL is commonly used with Laravel.
5. **Web Server**:
    - Laravel includes a built-in server to help you test your website locally (on your computer).

If you don’t have these tools yet, search for tutorials to install PHP, Composer, and MySQL. They’re free and widely used.

How to Install Laravel

Step 1: Install Composer

Composer is like an app store for Laravel. To install it:

1. Open your terminal (Command Prompt on Windows).
2. Type this command and press Enter:

3.
   ```
   curl -sS https://getcomposer.org/installer | php
   mv composer.phar /usr/local/bin/composer
   ```

   This downloads Composer to your computer.

Step 2: Create a New Laravel Project

Use Composer to create a new Laravel project. In your terminal, type:

```
composer create-project
--prefer-dist laravel/laravel
my-project
```

Replace `my-project` with the name of your project.

Step 3: Start Laravel

Go into your project folder and start Laravel's built-in server:

```
cd my-project
php artisan serve
```

This starts a web server. Open your browser and go to `http://localhost:8000`. You'll see the Laravel welcome page.

How Laravel Works (Step-by-Step)

Here are the key terms you'll encounter and what they mean:

1. Routes:

- Routes are like roadmaps. They tell Laravel what to do when someone visits a specific web address.
- Example in plain PHP:
- `if ($_SERVER['REQUEST_URI'] == '/') {`
- `    echo 'Hello, World!';`
  `}`

- In Laravel, you define routes in the `routes/web.php` file:
- `Route::get('/', function () {`
- `    return 'Hello, World!';`
  `});`

  This code tells Laravel to display "Hello, World!" when someone visits the homepage.

2. Controllers:

- Controllers handle the logic of your application. They decide what data to show and what happens when users interact with your website.
- Example in plain PHP:
- `if ($_SERVER['REQUEST_URI'] == '/')`
  `{`
- `    echo '<h1>Welcome to my`
  `website!</h1>';`
  `}`

- In Laravel, you create a controller using:

```
php artisan make:controller
MyController
```

Then, define logic in the controller and link it to a route:

```
public function showWelcome() {
    return 'Welcome to my Laravel
app!';
}
```

3. Views:

- Views are the HTML pages your users see. They separate the design (HTML) from the logic (PHP).
- Example in plain PHP:

```
echo '<h1>Welcome to my
website!</h1>';
```

- In Laravel, you create views in the `resources/views` folder:
- `<!-- File: welcome.blade.php -->`

```
<h1>Welcome to my Laravel
app!</h1>
```

Link it to a route:

```
Route::get('/', function () {
    return view('welcome');
});
```

4. Database:

- Laravel uses a database to store information, like user data or product details.
- In plain PHP:
- `$conn = new mysqli($servername, $username, $password, $dbname);`
- `$sql = "INSERT INTO tasks (name) VALUES ('Learn PHP')";`
  `$conn->query($sql);`

- In Laravel, you configure the database in the `.env` file:
- `DB_CONNECTION=mysql`
- `DB_HOST=127.0.0.1`
- `DB_PORT=3306`
- `DB_DATABASE=your_database_name`
- `DB_USERNAME=your_username`
  `DB_PASSWORD=your_password`

5. Migrations:

- Migrations are like version control for your database. They help you create or modify database tables easily.
- Example in Laravel:

```
php artisan make:migration
create_tasks_table
```

Add table columns:

```
Schema::create('tasks', function
(Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->timestamps();
});
```

Run the migration:

```
php artisan migrate
```

Building a Simple To-Do App

Let's create a simple to-do list to understand Laravel better.

Step 1: Define Routes

Add routes in `routes/web.php`:

```
Route::get('/tasks',
[TaskController::class, 'index']);
Route::post('/tasks',
[TaskController::class, 'store']);
```

Step 2: Create a Controller

Run this command:

```
php artisan make:controller
TaskController
```

In
`app/Http/Controllers/TaskController.php`,
add:

```
use App\Models\Task;

public function index() {
    $tasks = Task::all();
    return view('tasks.index',
compact('tasks'));
}

public function store(Request $request)
{
    Task::create(['name' =>
$request->name]);
    return redirect('/tasks');
}
```

Step 3: Create a View

Create `List</h1>`

```
    <form method="POST"
action="/tasks">
        @csrf
        <input type="text" name="name"
placeholder="New Task">
```

```
        <button
type="submit">Add</button>
    </form>
    <ul>
        @foreach ($tasks as $task)
            <li>{{
resources/views/tasks/index.blade.ph
p:
<!DOCTYPE html>
<html>
<head>
    <title>To-Do List</title>
</head>
<body>
    <h1>To-Do$task->name }}</li>
        @endforeach
    </ul>
</body>
</html>
```

Key Differences: Laravel vs Plain PHP

Feature	Plain PHP	Laravel
Routes	Manual `if` statements	Organized `web.php` file
Controllers	Mixed with other code	Separate logic in controllers

Views	HTML and PHP mixed	Blade templates
Database	Manual SQL queries	Models and migrations
Security	Must be implemented manually	Built-in tools (e.g., CSRF)

Learning Tips

1. **Start Small**: Build small projects to understand the basics.
2. **Use Laravel Docs**: The official documentation is clear and helpful.
3. **Practice Often**: The more you code, the better you'll understand.

Now you're ready to start building with Laravel! Take it one step at a time, and enjoy coding!